Monte Carlo Tree Search without Numerical Rewards

Tobias Joppen[1]

Abstract. In many problem settings, most notably in game playing, an agent receives a possibly delayed reward for its actions. Often, those rewards are handcrafted and not naturally given. It is hard to argue about good rewards and the performance of an agent often depends on the design of the reward signal. Creating unbiased rewards is a hard to impossible task: In particular, in domains where states by nature only have an ordinal ranking and where meaningful distance information between game state values is not available, a numerical reward signal is necessarily biased. We search for alternatives of numerical rewards in the context of artificial intelligence. Here, we take a look at Monte Carlo Tree Search (MCTS) and highlight a reoccurring problem concerning its use of numerical rewards and how non-numerical rewards can overcome this problem.

1 Introduction

In many real world problems an agent has to execute a sequence of actions to move from one state to another until a terminal state is reached. The main task is to find the best sequence from a given world state. How to define what *best sequence* means is easy on a theoretical level but often hard to do well in practice. In theory the best sequence maximizes the rewards the agent gets by executing the actions. In practice it is hard to design rewards such that agents perform well.

The main problem here are delayed rewards: An action may be very good and all follow up states lead to high rewards, but the reward of the action itself does not represent it. In fact, designing rewards without delayed rewards equals to mathematically solving the complete problem, what often is impossible due to complexity.

1.1 Monte Carlo Tree Search (MCTS)

Monte Carlo tree search (MCTS) is a method for approximating an optimal policy for sequential problems. It builds a partial search tree, which is more detailed where the rewards are high. MCTS spends less time evaluating less promising action sequences, but does not avoid them entirely in order to explore the state space. The algorithm iterates over four steps [2]: *Selection:* Starting from the root node which corresponds to start state, a *tree policy* traverses to deeper nodes, until a state with unvisited successor states is reached. *Expansion:* One successor state is added to the tree. *Simulation:* Starting from the new state, a so-called *rollout* is performed, i.e., random actions are played until a terminal state is reached or a depth limit is exceeded. *Backpropagation:* The reward of the last state of the simulation is backed up through the selected nodes in tree. For vanilla MCTS, the tree policy chooses that action, which maximizes the average future reward per node plus an exploration bonus favoring rarely visited nodes to find new solutions (UCT formula, see [5]). The average future reward is nothing but the average of the backed up reward.

[1] Technical University Darmstadt, email: tjoppen@ke.tu-darmstadt.de

MCTS works quite good and converges towards the optimal action sequence. But given a complex problem, it often takes too long for MCTS to find any suitable solution. MCTS is an anytime algorithm and thus it can be stopped early by returning the current best action (sequence). To improve the performance in this case, the rollouts of MCTS often get limited by a maximal number of steps. This way, action estimations represent the rewards in near future instead of failing to estimate the overall value of actions, which is too noisy since random sampling to an unbounded depth is expensive with a high variance.

1.2 Problematic Example

A reoccurring problem of MCTS is its behavior in case of danger: As an example we look at a generic platform game such as Super Mario, where an agent has to jump over deadly gaps to eventually reach the goal at the right. Dying is very bad, and the more the agent proceeds to the right, the better. To guide the agent to the goal, the rewards are designed such that moving right grants a positive reward, moving left a negative, reaching the goal a very high and falling in a gap and die a very negative one. This kind of reward shaping is very common.

The problem occurs by comparing the actions *jump* and *stand still* : jumping either leads to a better state than before because the agent proceeded to the right by successfully jumping a gap, or to the worst possible state (*death*) in case the jump attempt failed. Standing still, on the other hand, safely avoids death, but will never advance to a better game state. MCTS averages the obtained rewards gained by experience, which lets it often choose the safer action and therefore not progress in the game, because the (few) experiences ending with its death pull down the average reward of *jump* below the mediocre but steady reward of standing still. Because of this, the behavior of MCTS has also been called *cowardly* in the literature [3].

2 Our Approaches

In this section, we analyze the problem stated above and show that a preference-based view changes the agents behavior. Thereafter we sketch our two approaches following this path: Preference-Based MCTS and Ordinal MCTS and show problems and future work.

2.1 Problem Revisited

In Figure 1 an example reward distribution of the two actions *jump*(star) and *stand still*(circle) are shown. MCTS averages the good states that reached the other side of the gap with the failed approaches where the game was lost. This average lies below the average of the safe stand still action. That is why MCTS favors the safe action. To overcome this problem, the literature often modifies rewards or adds a bias the algorithm [3].

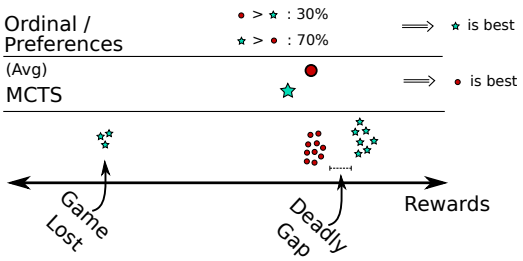

Figure 1. Two actions with different distributions.

Non numeric. We propose a different approach: The numerical distances between the reward values of *lost game*, *left* and *right of gap* to a high degree are chosen arbitrary. The point is those distances matter: The relative distances between those three states define whether MCTS favors *jump* over *stand still* or vice versa.

As opposed to the distances between rewards, the ordering of those rewards is nonarbitrary: It is hard to argue against *lost game* $\prec$ *left of gap* $\prec$ *right of gap*. Given only this information and the example of Figure 1, *jump* beats *stand still* in 70% and thus is the better action. But it is not possible to run MCTS on preferences only: you can not compute averages and have to directly compare a reward to another to get any useful information. In the following we present two MCTS modifications to follow this approach.

2.2 Preference-Based MCTS

The idea of Preference-Based MCTS (PB-MCTS) is to always compare two actions instead of only evaluating one. Instead of choosing a path from root to a leaf node, PB-MCTS selects a binary subtree. Instead of using the UCT formula, which has its roots in multi-armed bandit setting, we use the RUCB formula, which has its origin in the dueling bandit setting. This way we select two actions in each node, which rewards will be compared to get a preference information. The RUCB formula uses this information in later iterations to choose new actions to compare. This algorithm can either be seen as a natural extension of RUCB to tree search, just as UCT is the tree extension of UCB or can be seen as the preference version of MCTS, just as RUCB is a preference version of UCB.

Drawbacks and Future Work. Spanning a binary tree leads to an exponential number of rollouts in each iteration. Only few of those rollouts corresponds to good actions and the nice asymmetric growth of the search tree gets restricted. Further, given n actions in a node vanilla MCTS needs n iterations to have a reasonable estimate about the values of those actions. In comparison PB-MCTS, which relies on RUCB that does not corporate transitivity between actions, needs a comparison of each action pair to have a first fair estimate.

Those problems may be tackled with alternatives to RUCB incorporating transitivity (the tree structure even does not support non transitive problems) and by using pruning. An alternative to PB-MCTS that does not have those drawbacks is Ordinal MCTS.

2.3 Ordinal MCTS

Vanilla MCTS can not learn from a single rollout using an ordinal reward because it can not average ordinal rewards. The idea of Ordinal MCTS (O-MCTS) is to store past rewards in each node. This way a new reward can be compared to rewards of other actions and thus one can get a qualitative information about the current reward.

In comparison to PB-MCTS, O-MCTS does not change anything at the iteration steps except of the way rewards get stored and handled. Given the possible ordinal rewards O, a naïve implementation of O-MCTS is to have an array c of length $|O|$ in each node and store the reward distribution perceived. In backpropagation, simply increase $c[o]$ in each selected node. The choice of which action to choose in the selection step is as follows: Given an array c_n for each child node n the Borda Score [1] can be used to compute the probability of each node/action to win against a random competitor. This way one can calculate a value for each action which can be used as a substitution for the average value used in MCTS. Now, using this value, the UCT formula can be used to choose the node/action to select.

Drawbacks and Future Work. Storing an array in each node and computing the Borda Score takes more computational time and memory than the running average used in vanilla MCTS. One could use faster data structures than arrays and the Borda Score can be updated iteratively which reduces time complexity, but O-MCTS still needs more resources than MCTS. Additionally, the Borda Score is only one possibility to compute a action value given reward distributions of multiple actions. It is open for further research to try alternatives.

2.4 Experiments and Results

We have done experiments using game playing. Using the 8-puzzle, a low action space domain, PB-MCTS performs comparable to MCTS [4]. A more detailed experiment has been done using the General Video Game Framework (GVGAI, [6]). Here the action and state space is bigger and PB-MCTS performs worse than MCTS. O-MCTS, which has not been tested on the 8-puzzle, performs significantly better than MCTS here (Paper is under review).

3 Conclusion

We shortly motivated and introduced non numerical MCTS variants preference-based MCTS and ordinal MCTS. Using a preference-based view on rewards those algorithms do not behave as shy as vanilla MCTS by maximizing winning, not average rewards. Also we show drawbacks of both algorithms and point out possible future work.

Acknowledgments This work was supported by the German Research Foundation (DFG project number FU 580/10).

REFERENCES

[1] Duncan Black, 'Partial justification of the Borda count', *Public Choice*, **28**(1), 1–15, (1976).
[2] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, 'A survey of Monte Carlo tree search methods', *IEEE Transactions on Computational Intelligence and AI in Games*, **4**(1), 1–43, (2012).
[3] Emil Juul Jacobsen, Rasmus Greve, and Julian Togelius, 'Monte Mario: Platforming with MCTS', in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 293–300. ACM, (2014).
[4] Tobias Joppen, Christian Wirth, and Johannes Fürnkranz, 'Preference-based Monte Carlo tree search', *arXiv preprint arXiv:1807.06286*, (2018).
[5] Levente Kocsis and Csaba Szepesvári, 'Bandit based Monte-Carlo planning', in *Proceedings of the 17th European Conference on Machine Learning (ECML-06)*, pp. 282–293, (2006).
[6] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Simon M Lucas, and Tom Schaul, 'General video game AI: Competition, challenges and opportunities', in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pp. 4335–4337, (2016).